STLmc: Robust STL Model Checking of Hybrid Systems using SMT

Geunyeol Yu, Jia Lee, and Kyungmin Bae[0000−0002−6430−5175]

Pohang University of Science and Technology, Pohang, Korea

Abstract. a
a
a
a
a
a
a
a
a
a

1 Introduction

STL: What is it and why is it important?
a
a
a
a
a
a
a
a

STL analysis methods: also introduce STL model checking
a
a
a
a
a
a
a

Challenges

- robust model checking: important for hybrid systems... how?
- analyzing hybrid systems with nonlinear ODEs (not addressed before, and just using dReal is infeasible)
- no tool supporting the above features, together with user-frendly interface, has been developed before

This paper: how the above challenges have been addressed

a
a
a
a
a
a
a
a

Key features and algorithms

a
a
a
a
a
a
a
a

a
a
a
a
a

Evaluation, and other notes like a tool webpage

a
a
a
a
a
a
a

2 Background

2.1 Hybrid Automata

Hybrid automata are widely used for formalizing cyber-physical systems (CPSs) that exhibit both discrete and continuous behaviors. In a hybrid automaton H, a set of *modes* Q specifies discrete states, and a finite set of real-valued *variables* $X = \{x_1, ..., x_l\}$ specifies continuous states. A state of H is a pair $\langle q, \vec{v} \rangle \in Q \times \mathbb{R}^l$ of mode q and real-valued vector $\vec{v}$. An initial condition $init(\langle q, \vec{v} \rangle)$ defines a set of initial states. An invariant condition $inv(\langle q, \vec{v} \rangle)$ defines a set of valid states. A flow condition $flow(\langle q, \vec{v} \rangle, t, \langle q, \vec{v}_t \rangle)$ defines a *continuous evolution* of variables X from a value $\vec{v}$ to $\vec{v}_t$ over duration t in mode q. A jump condition $jump(\langle q, \vec{v} \rangle, \langle q', \vec{v'} \rangle)$ defines a discrete transition from state $\langle q, \vec{v} \rangle$ to $\langle q', \vec{v'} \rangle$.

Definition 1. *A hybrid automaton is a tuple* $H = (Q, X, init, inv, jump, flow)$.

A *signal* σ represents a continuous execution of a hybrid automaton H, given by a function $[0, \tau) \to Q \times \mathbb{R}^l$ with a time bound $\tau > 0$. A signal σ is called *bounded* if $\tau < \infty$. A signal σ is called a *trajectory* of a hybrid automaton H, written $\sigma \in H$, if σ describes a valid behavior of H.

Definition 2. *For a hybrid automaton* H*, a signal* $\sigma : [0, \tau) \to Q \times \mathbb{R}^l$ *is a trajectory of* H*, written* $\sigma \in H$*, if there exist sequences of modes* $q_1, q_2, q_3, ...$ *and of times* $0 = t_0 < t_1 < ... < \tau$ *such that:*

- *the initial condition holds at time* t_0*: i.e.,* $init(\sigma(t_0))$ *holds;*
- *for* $i \geq 1$*, the values of* X *change from* $\sigma(t_{i-1})$ *for time* $t_i - t_{i-1}$ *by the flow condition, satisfying the invariant condition: i.e., for any* $t \in [t_{i-1}, t_i)$*:*

$$flow(\sigma(t_{i-1}), t - t_{i-1}, \sigma(t)) \quad and \quad inv(\sigma(t))$$

- *for* $i \geq 1$*, a discrete jump happens at time* t_i*: i.e., for some* $s_i \in Q \times \mathbb{R}^l$*:*

$$flow(\sigma(t_{i-1}), t_i - t_{i-1}, s_i) \quad and \quad jump(s_i, \sigma(t_i))$$

K: the previous definition was wrong. Please check the corrected version.

Example 1. There are two rooms connected by an open door. The temperature x_i of each room $i \in \{0, 1\}$ is controlled by each thermostat, depending on the heater's mode $q_i \in \{\mathsf{On}, \mathsf{Off}\}$ and the other room's temperature. The continuous dynamics of x_i can be given as ODEs as follows [3, 14]:

$$\dot{x_i} = \begin{cases} K_i(h_i - (c_i x_i - d_i x_{1-i})) & (\mathsf{On}) \\ -K_i(c_i x_i - d_i x_{1-i}) & (\mathsf{Off}), \end{cases}$$

where K_i, h_i, c_i, d_i are constants depending on the size of the room, the heater's power, and the size of the door. Figure 1 shows a hybrid automaton of our thermostat controllers. Initially, both heaters are off and the temperatures are between 18 and 22. The jumps between modes then define a control logic to keep the temperatures within a certain range with only one heater on.

Geunyeol: Update the hybrid automaton

2.2 Signal Temporal Logic

Signal temporal logic (STL) can be used to specify properties of CPS. The syntax of STL over a set of state propositions Π is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

where $p \in \Pi$ denotes state propositions and $I \subseteq \mathbb{R}_{\geq 0}$ denotes time intervals of nonnegative real numbers.Examples of state propositions include relational expressions of the form $f(\vec{x}) \geq 0$ with a real-valued function $f : \mathbb{R}^l \to \mathbb{R}$. Other common boolean and temporal operators can be derived by equivalences: e.g.,

$$\varphi \vee \varphi' \equiv \neg(\neg\varphi \wedge \neg\varphi'), \quad \Diamond_I \varphi \equiv \top \, \mathbf{U}_I \, \varphi, \quad \varphi \, \mathbf{R}_I \, \varphi' \equiv \neg((\neg\varphi) \, \mathbf{U}_I (\neg\varphi'))$$

K: This is not used in the paper. Removed.

K: $sub(\varphi)$ and $prop(\varphi)$ look not used. Removed

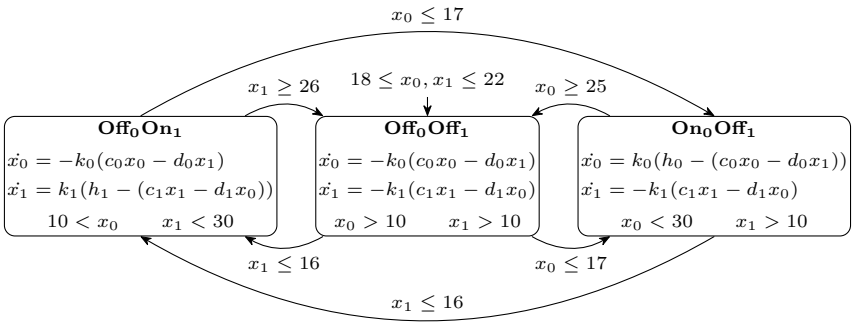


Fig. 1: A hybrid automaton of two thermostats.

Example 2. For Example 1, we consider the following STL formulas:

- $\square_{[2,4]}\,(x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}\, x_0 - x_1 \leq -3)$
- $\Diamond_{[0,3]}\,(x_0 \geq 13\ \mathbf{U}_{[0,\infty)}\ x_1 \leq 22)$
- $\square_{[0,10]}\,(x_0 > 23\ \mathbf{R}_{[0,\infty)}\ x_0 - x_1 \geq 4)$

K: Some notations should be introduced

Geunyeol: update right open to close interval and update formula 2

The Minkowski sum of two intervals I and J is denoted by $I + J$. E.g., $[a, b] + [c, d] = [a + c, b + d]$. For a singular interval $\{t\}$, the set $\{t\} + I$ is often written as $t + I$. We write $\sup_{t\in I} g(t)$ and $\inf_{t\in I} g(t)$ to denote the least upper bound and the greatest lower bound of the set $\{g(t) \mid t \in I\}$, respectively.

We consider a quantitative semantics of STL based on *robustness degrees* [9]. The semantics of a proposition p is specified as a function $p : Q \times \mathbb{R}^l \to \overline{\mathbb{R}}$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$, that assigns to a state of H the degree to which p is true. For a state proposition of the form $f(\vec{x}) \geq 0$, the robustness degree is the value of $f(\vec{x})$ at a given state. E.g., the robustness degree of proposition $x_0 - x_1 \geq 4$ in Example 2 is the value of $x_0 - x_1 - 4$ at a given state.

K: The previous definition was wrong, because time bound τ is not considered for $\mathbf{U}_I$. Check the corrected definition

The robustness degree of an STL formula can be defined inductively [9]. We explicitly take into account a time bound τ of a signal.

Definition 3. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the robustness degree $\rho_\tau(\varphi, \sigma, t) \in \overline{\mathbb{R}}$ is defined inductively by:*

$$\begin{aligned}
\rho_\tau(p, \sigma, t) &= p(\sigma(t)) \\
\rho_\tau(\neg\varphi, \sigma, t) &= -\rho_\tau(\varphi, \sigma, t) \\
\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) &= \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \\
\rho_\tau(\varphi_1\ \mathbf{U}_I\ \varphi_2, \sigma, t) &= \sup_{t' \in (t+I)\cap[0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t''))
\end{aligned}$$

The robust STL model checking problem is to check whether an STL formula φ always has a greater robustness degree than a given threshold $\epsilon > 0$ for any possible trajectory of a hybrid automaton H.

Definition 4 (Robust STL Model Checking). *Given a hybrid automaton H and a bound $\tau > 0$, an STL formula φ is satisfied at time t in H with respect to a robustness threshold $\epsilon > 0$ iff for every trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$.*

2.3 STL Boolean Semantic

We also introduce the Boolean semantics of STL, which was originally proposed as the semantics of STL in [17]. We consider the following definition from [16] that explicitly considers a time bound $\tau > 0$.

Definition 5. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the satisfaction of φ, denoted by $\sigma, t \models_\tau \varphi$ is inductively defined by:*

$$
\begin{array}{ll}
\sigma, t \models_\tau p & \text{iff } t < \tau \text{ and } p(\sigma(t)) = \top \\
\sigma, t \models_\tau \neg\varphi & \text{iff } \sigma, t \not\models_\tau \varphi \\
\sigma, t \models_\tau \varphi_1 \wedge \varphi_2 & \text{iff } \sigma, t \models_\tau \varphi_1 \text{ and } \sigma, t \models_\tau \varphi_2 \\
\sigma, t \models_\tau \varphi_1 \, \mathbf{U}_I \, \varphi_2 & \text{iff } \exists t' \in (t + I) \cap [0, \tau).\ \sigma, t' \models_\tau \varphi_2,\ \forall t'' \in [t, t').\ \sigma, t'' \models_\tau \varphi_1
\end{array}
$$

The following theorem shows the obvious relationship between the Boolean semantics and the quantitative semantics. If the robustness degree of φ is strictly positive (resp., negative), then φ is true (resp., false).

K: A subject sentence was missing.

Theorem 1. *[9] For an STL formula φ, a signal σ, and a time $t \in [0, \tau)$, $\rho_\tau(\varphi, \sigma, t) > 0$ implies $\sigma, t \models_\tau \varphi$, and $\rho_\tau(\varphi, \sigma, t) < 0$ implies $\sigma, t \not\models_\tau \varphi$.*

3 The STLMC Tool

This section introduces the STLMC tool. The input format describes a hybrid automaton and its STL properties. Our tool then performs STL model checking with respect to a given robustness threshold $\epsilon > 0$ and two bound parameters $\tau > 0$ (a time bound) and $k \in \mathbb{N}$ (a discrete bound). STLMC also provides a visualization script to give an intuitive description of counterexamples.

3.1 Input Format

The input format of STLMC consists of four sections: (i) variable declarations, (ii) mode declarations, which include mode, flow, jump, and invariant conditions, (iii) initial condition declarations, and (iv) property declarations.

Variable Declarations. STLMC uses mode and continuous variables to specify discrete and continuous states of a hybrid automaton. *Mode variables* define a set of discrete modes Q, and *continuous variables* define a set of real-valued variables X.[1] We can also declare named constants whose values do not change.

Mode variables are declared with one of three types: `bool` variables with `true` and `false` values, `int` variables with integer values, and `real` variables with real values. E.g., the following declares a `bool` variable `b` and an `int` variable `i`:

[1] An assignment to mode and continuous variables corresponds to a single state.

```
bool b;    int i;
```

Continuous variables are declared with domain intervals. Domains can be any intervals of real numbers, including open, closed, and half-open intervals, E.g., the following declares two continuous variable `x` and `y`:

```
[0, 50] x;    (-1.1, 1) y;
```

Finally, constants are introduced with the `const` keyword. Constants can have Boolean, integer, or rational values. For example, the following declares a constant `k1` with a rational value 0.015:

```
const k1 = 0.015;
```

Mode Declarations. In STLMC, *mode blocks* define mode, jump, invariant, and flow conditions for a group of modes in a hybrid automaton. Multiple model blocks can be declared, and each mode block consists of four components:

```
{
    mode: ...
    inv:  ...
    flow: ...
    jump: ...
}
```

A `mode` component contains a semicolon-separated set of Boolean conditions over mode variables. The conjunction of these conditions represents a set of modes.[2] E.g., the following represents a set of two modes $\{(true, 1), (true, 2)\}$, provided there are two mode variables `b` (of `bool` type) and `i` (of `int` type):

K: is this footnote below true?

```
b = true;    i > 0;    i < 3;
```

An `inv` component contains a semicolon-separated set of Boolean formulas over continuous variables. The conjunction of these conditions then represents the invariant condition for each mode in the mode block. For example, the following declares an invariant condition for two continuous variables `x` and `y`:

```
x < 30;    y > - 0.5;
```

A `flow` component contains either a system of ordinary differential equations (ODEs) or a closed-form solution of ODEs. In STLMC, a system of ODEs over continuous variables $x_1, \ldots, x_n$ is written as a semicolon-separated equation of the following form, where e_i denotes an expression over $x_1, \ldots, x_n$:

[2] We assume that the mode conditions of different mode blocks cannot be satisfied at the same time, which can be automatically detected by our tool.

```
d/dt[x1] = e1(x1,...,xn) ;
...
d/dt[xn] = en(x1,...,xn) ;
```

A closed-form solution of ODEs is written as a set of continuous functions, parameterized by a time variable `t` and the initial values $x_1(0), ..., x_n(0)$:

```
x1(t) = e1(t, x1(0),...,xn(0)) ;
...
xn(t) = en(t, x1(0),...,xn(0)) ;
```

A `jump` component contains a set of jump conditions *guard* `=>` *reset*, where *guard* and *reset* are Boolean conditions over mode and continuous variables. We use "primed" variables to denote states after jumps have occurred. E.g., the following defines a jump with four variables `b`, `i`, `x`, and `y` (declared above):

```
(and (i = 1) (x > 10)) => (and (b' = false) (i = 3) (x' = x) (y' = 0));
```

Initial Condition Declarations. In the `init` section, an initial condition is declared as a set of Boolean formulas over mode and continuous variables. Similarly, the conjunction of these conditions represents a set of initial modes. E.g., the following shows an initial condition with variables `b`, `i`, `x`, and `y`:

```
init: (and (not b) (i = 0) (19.9 <= x) (y = 0));
```

Property Declarations. In the `goal` section, STL properties are declared with labels. State propositions are arithmetic and relational expressions over mode and continuous variables, and labels can be omitted. For example, the following declares the first STL formula in Example 2 with label `f1`:

```
f1: [][2, 4]((x0 - x1 >= 4) -> <>[3, 10] (x0 - x1 <= -3));
```

To make it easy to write repeated propositions, "named" state propositions can be declared in the `proposition` section. For example, the above STL formula can be rewritten using two propositions `vl` and `vr` as follows:

K: It is odd that named "subformulas" are written in the proposition section. Corrected.

```
proposition:
[vl]: x0 - x1 >= 4;
[vr]: x0 - x1 <= -3;

goal:
[f1]: [][0, 4](vl -> <>[0, 10] vr);
```

An Example. Figure 2 shows the input model for the hybrid automaton in Example 1 and the STL properties in Example 2. There are two mode variables `on0` and `on1` to denote the heaters's modes, and two continuous variables `x0` and `x1` to denote the temperatures of the rooms. We use two propositions `p1` and `p2` to denote common state propositions in the STL formulas in `goal`. **Geunyeol: update `x1` and `y2` to `x0` and `x1`**

```
int on0;                int on1;
[10, 35] x0;            [10, 35] x1;
const k0 = 0.015;       const k1 = 0.045;
const h0 = 100;   const h1 = 200;
const c0 = 0.98;        const c1 = 0.97;
const d0 = 0.01;        const d1 = 0.03;

{ mode: on0 = 0;        on1 = 0;
  inv:  x0 > 10;        x1 > 10;
  flow:
    d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
    d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump:
    x0 <= 17 => (and (on0' = 1) (on1' = on1)
                     (x0' = x0) (x1' = x1));
    x1 <= 16 => (and (on1' = 1) (on0' = on0)
                     (x0' = x0) (x1' = x1));
}
{ mode: on0 = 0;        on1 = 1;
  inv:  10 < x0;        x1 < 30;
  flow:
    d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
    d/dt[x1] = k1 * (h1 - (c1 * x1 - d1 * x0));
  jump:
    x0 <= 17 => (and (on0' = 1) (on1' = 0)
                     (x0' = x0) (x1' = x1));
    x1 >= 26 => (and (on1' = 0) (on0' = on0)
                     (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;        on1 = 0;
  inv:  x0 < 30;        x1 > 10;
  flow:
    d/dt[x0] = k0 * (h0 - (c0 * x0 - d0 * x1));
    d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump:
    x1 <= 16 => (and (on0' = 0) (on1' = 1)
                     (x0' = x0) (x1' = x1));
    x0 >= 25 => (and (on0' = 0) (on1' = on1)
                     (x0' = x0) (x1' = x1));
}
init: on0 = 0;  18 <= x0;  x0 <= 22;
      on1 = 0;  18 <= x1;  x1 <= 22;
propositions:
  [p1]: x0 - x1 >= 4;    [p2]: x0 - x1 <= -3;
goal:
  [f1]: [][2, 4](p1 -> <>[3, 10] p2);
  [f2]: <>[0, 3](x0 >= 13 U[0, inf) x1 <= 22);
  [f3]: [][0,10] (x0 > 23 R[0,inf) p1);
```

Fig. 2: An input model for Example 1

3.2 Command Line Options

K: Some options are changed, and less important options are removed from the table.

The STLmc tool provides a command-line interface with various command line options, summarized in Table 1. A discrete bound N limits the number of mode changes and the number of *variable points*—at which the truth value of some STL subformula changes—in trajectories. A time horizon T limits the maximum time duration of single modes in trajectories. The options `-two-phase-opt` and `-parallel` enable the (parallelized) two-phase optimization in Sec. 4.3. When the option `-visualize` is set, extra data for visualization is generated.

We can choose different SMT solvers using the `-solver` option. The STLmc tool currently supports three SMT solvers: Z3 [8], Yices2 [10] and dReal [13]. The underlying solver is chosen depending on the flow conditions of a hybrid automaton. Z3 and Yices2 can deal with linear and polynomial functions, and dReal can deal with nonlinear ODEs—which is undecidable in general for hybrid automata—approximately up to a given precision $\delta > 0$.

Option	Explanation	Default
`-bound <`N`>`	a discrete bound N	-
`-time-bound <`τ`>`	a time bound τ	-
`-time-horizon <`T`>`	a mode duration bound T	τ
`-threshold <`ϵ`>`	a robustness threshold ϵ	0.01
`-goal`	the list of STL goals to be analyzed	all
`-two-phase-opt`	use the two-phase optimization	disabled
`-parallel`	parallelize the two-phase optimization	disabled
`-visualize`	generate extra visualization data	disabled
`-solver <Solver>`	an SMT solver to be used (`z3`/`yices`/`dreal`)	`z3`
`-precision <`δ`>`	a precision parameter for `dreal`	0.001

Table 1: The command line options of STLmc.

Example 3. Consider the input model in Fig. 2 (`therm.model`). The following command found a counterexample of the formula `f1` at bound 3 with respect to robustness threshold $\epsilon = 2$ in 83 seconds.

```
$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 2 \
         -goal f1 -solver dreal -two-phase-opt -parallel -visualize
goal: []_[2.0,4.0] (p1 -> (<>_[3.0,10.0] p2))
result: counterexample found (bound 3)
running time: 83.27295 seconds
```

The following command verifies the formula `f2` up to bounds $N = 5$ and $\tau = 30$ with respect to robustness threshold $\epsilon = 1$ in 796 seconds.

```
$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 1 \
         -goal f2 -solver dreal -two-phase-opt -parallel
goal : (<>_[0.0,3.0] ((x0 >= 13) U_[0.0,inf) (x1 <= 22)))
result : True
running time 795.99192 seconds
```

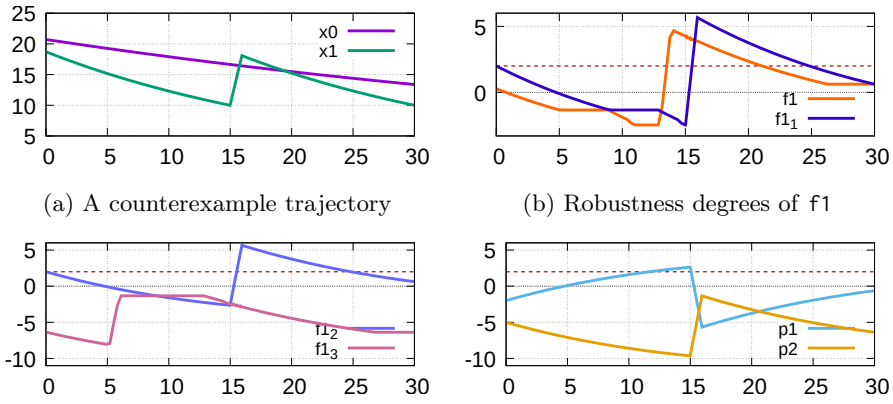
Geunyeol: update commands

3.3 Visualization

The STLmc tool provides a script to visualize counterexamples for robust STL model checking. The visualization command can generate HTML files or PDF images that contain graphs to represent counterexample trajectories of a hybrid automaton or robustness degrees for each subformula of an STL formula φ.

Example 4. Consider the model checking command for `f1` in Example 3, where `-visualize` is enabled. The following command generate a PDF image.

```
./stlmc-vis ./therm.model -output pdf
```



(a) A counterexample trajectory

(b) Robustness degrees of `f1`

(c) Robustness degrees of `f1_1` and `f1_2`

(d) Robustness degrees of `p1` and `p2`

Fig. 3: Visualization of a counterexample for Example 4, where horizontal dotted lines denote the given threshold $\epsilon = 2$.

Figure 3 shows the (slightly adapted) visualization graphs generated by the above command, where $f1 = \square_{[2,4]}\left((x_0 - x_1 \geq 4) \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3)\right)$, $f1_1 = (x_0 - x_1 \geq 4) \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3)$, and:

$$f1_2 = \neg(x_0 - x_1 \geq 4) \qquad p_1 = x_0 - x_1 \geq 4$$
$$f1_3 = \Diamond_{[3,10]}(x_0 - x_1 \leq -3) \qquad p_2 = x_0 - x_1 \leq -3$$

In the graph, the robustness degree of $f1$ is less than ϵ at time 0, because the robustness degree of its subformula $f1_1$ goes below ϵ in the time interval $[2, 4]$, which is because the maximum of the robustness degrees of $f1_2$ and $f1_3$ is less than ϵ in $[2, 4]$. The robustness degree of $f1_3$ is less than ϵ in $[2, 4]$, because the robustness degree of p_2 is less than ϵ in the interval $[5, 14] = [2, 4] + [3, 10]$.

Geunyeol: update graph image

4 Algorithms and Implementation

This section explains the algorithms and implementation of our STLMC tool. Section 4.1 explains how to reduce the robust STL model checking problem to the Boolean STL model checking problem. Section 4.2 summarizes a Boolean STL model checking algorithm, introduced in [4,16]. Section 4.3 explains a two-phase solving optimization to improve the performance for ODEs. Finally, Section 4.4 presents the tool's architecture and implementation.

4.1 Reduction to Boolean STL Model Checking

As usual for model checking, robust STL model checking is equivalent to finding a counterexample. We can easily see that an STL formula φ is not satisfied in

a hybrid automata H with respect to a robustness threshold $\epsilon > 0$ iff there is a trajectory that the robustness degree of $\neg\varphi$ is greater than or equal to $-\epsilon$.[3]

Corollary 1. *For a hybrid automaton H, a time $t \in [0, \tau)$, and a robustness threshold $\epsilon > 0$, an STL formula φ is not satisfied at t in H with respect to ϵ iff there exists a trajectory $\sigma \in H$ such that $\rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.*

Our goal is to reduce finding a counterexample of robust STL model checking into finding a counterexample of Boolean STL model checking. Let us first a simple case, say, a state proposition $x \geq 0$. Its robust model checking with respect ϵ is equivalent to finding a counterexample $\sigma \in H$ with $\rho_\tau(-x > 0, \sigma, t) \geq -\epsilon$ by Corollary 1, which is equivalent to $\rho_\tau(-x > -\epsilon, \sigma, t) \geq 0$. Notice that $-x > -\epsilon$ can be obtained by *weakening* $-x \geq 0$ by ϵ.

The notions of ϵ-weakening ϵ-strengthening are first introduced in [12] for first-order formulas. In this paper, we define the notions of ϵ-weakening and ϵ-strengthening for STL formulas as follows.

Definition 6. *The ϵ-strengthening and ϵ-weakening of an STL formula φ for $\epsilon > 0$, denoted by $\varphi^{+\epsilon}$ and $\varphi^{-\epsilon}$, respectively, are defined inductively as follows:*

- $p^{+\epsilon} \equiv \rho + \epsilon$
- $(\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon})$
- $(\varphi_1 \wedge \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \wedge \varphi_2^{+\epsilon}$
- $(\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \,\mathbf{U}_I\, \varphi_2^{+\epsilon}$
- $p^{-\epsilon} \equiv \rho - \epsilon$
- $(\neg\varphi)^{-\epsilon} \equiv \neg(\varphi^{+\epsilon})$
- $(\varphi_1 \wedge \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \wedge \varphi_2^{-\epsilon}$
- $(\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \,\mathbf{U}_I\, \varphi_2^{-\epsilon}$

where $\rho + \epsilon$ is defined by $\rho + \epsilon(\langle q, \vec{v}\rangle) = \rho(\langle q, \vec{v}\rangle) + \epsilon$, and $\rho - \epsilon$ is defined by $\rho - \epsilon(\langle q, \vec{v}\rangle) = \rho(\langle q, \vec{v}\rangle) - \epsilon$, where $\langle q, \vec{v}\rangle$ is a state of a hybrid automaton.

The following lemma states that the relationship between the satisfaction of an STL formula in robustness STL model chekcing and the satisfactions of $\varphi^{+\epsilon}$ and $\varphi^{-\epsilon}$ in STL model checking.

Lemma 1. *For a signal σ and an STL property φ:*

(1) $\rho(\varphi, \sigma, t) > \epsilon \;\Rightarrow\; \sigma, t \models_\tau \varphi^{+\epsilon}$
(2) $\sigma, t \models_\tau \varphi^{-\epsilon} \;\Rightarrow\; \rho(\varphi, \sigma, t) \geq -\epsilon$
(3) $\sigma, t \models_\tau \varphi^{+\epsilon} \;\Rightarrow\; \rho(\varphi, \sigma, t) \geq \epsilon$
(4) $\rho(\varphi, \sigma, t) > -\epsilon \;\Rightarrow\; \sigma, t \models_\tau \varphi^{-\epsilon}$

According to above lemmas, we can reduce finding a counterexample for robust STL model checking of an STL formula φ into finding a counterexample for Boolean STL model checking of the ϵ-weakening of $\neg\varphi$, i.e., $(\neg\varphi)^{-\epsilon}$.

Theorem 2. *For a hybrid automaton H, an STL formula φ, a time t, a time bound τ, and an arbitrary positive real value ϵ, the following properties are hold:*

(1) $\exists\sigma \in H.\ \sigma, t \models_\tau (\neg\varphi)^{-\epsilon} \implies \exists\sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$

[3] By definition, φ is satisfied at t in H with respect to ϵ iff for any trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$ holds. Notice that $\neg(\forall\sigma \in H.\ \rho_\tau(\varphi, \sigma, t) > \epsilon)$ iff $\exists\sigma \in H.\ \rho_\tau(\varphi, \sigma, t) \leq \epsilon$ iff $\exists\sigma \in H.\ -\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$ iff $\exists\sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.

(2) $\forall \sigma \in H,\, \sigma, t \not\models_\tau (\neg\varphi)^{-\epsilon} \implies \forall \sigma \in H,\, \rho_\tau(\varphi, \sigma, t) \geq \epsilon$

As a consequence, a counterexample for Boolean STL model checking of $(\neg\varphi)^{-\epsilon}$ is also a counterexample for robust STL model checking of φ with respect to ϵ. If there is no counterexample of $(\neg\varphi)^{-\epsilon}$, then it is guaranteed that the robustness degree of φ is always greater than or equal to ϵ. Therefore, for any robustness threshold $0 < \epsilon' < \epsilon$, φ is satisfied at t in H with respect to ϵ'. It is worth noting that φ may not be satisfied with respect to ϵ itself.

4.2 Boolean STL Bounded Model Checking

For Boolean STL model checking, there exist *refutationally complete* bounded model checking algorithms [4, 16]. There are two bound parameters for these algorithms: a *time bound* τ that restricts a time domain of a hybrid automaton, and a *discret bound* N that restricts the number of mode changes and the number of variable points at which the truth value of some STL subformula changes. There exists an SMT encoding $\Psi^{n,\tau}_{H,(\neg\varphi)}$, which is satisfiable if there exists a counterexample trajectory of H up to bounds N and τ:

Theorem 3. *[4, 16] There exists a trajectory $\sigma \in H$ with at most N variable points and mode changes such that $\sigma, t \not\models_\tau \varphi$ iff $\Psi^{N,\tau}_{H,(\neg\varphi)}$ is satisfiable.*

The satisfiability of $\Psi^{N,\tau}_{H,(\neg\varphi)}$ can be (approximately) determined using an SMT solver. When the flow conditions are linear or polynomial, the satisfiability can be precisely determined using Z3 [8] and Yices2 [10]. For hybrid automata with ODE dynamics, the satisfiability is undecidable in general, but a solver like dReal [13] can approximately determine its satisfiability.

4.3 Two-Phase Solving Optimization

The computation cost of dReal for ODEs is highly expensive. To reduce the complexity, we propose a two-phase SMT solving algorithm, summariazed in Alg. 1, in a similar way to the lazy SMT solving approach [20]:

1. We abstract the flow and invariant conditions from the encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ using Boolean variables. We then enumerate a satisfiable assignment using an SMT solver to obtain a possible *scenario*, given by a conjunction of literals, without taking into account the continuous dynamics.
2. For each conjunction of literals enumerated, we check the satisfiability using dReal considering the flow and invariant conditions. If any conjunction is satisfiable, we obtain a counterexample. If all conjunctions are unsatisfiable by dReal, it is guaranteed that there is no counterexample.

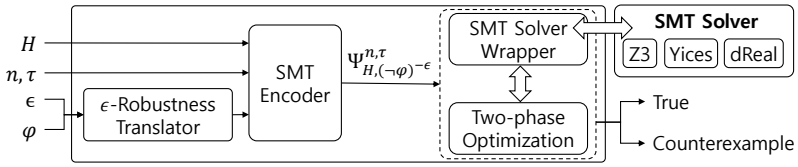


Fig. 4: The STLMC architecture

Algorithm 1: Two-Phase SMT Solving Algorithm

Input: Hybrid automaton H, STL formula φ, threshold ϵ, bounds τ and N

for $k = 1$ **to** N **do**
  $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}} \leftarrow$ Encoding of STL model checking at bound k;
  $\overline{\Psi}^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}} \leftarrow$ Abstraction of $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ without flow and inv
  while *checkSat*($\overline{\Psi}^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$) *is Sat* **do**
    $\pi \leftarrow$ a possible scenario based on the satisfying assignment
    $\pi_{\forall,\int} \leftarrow$ the conjunction of literals with flow and inv obtained from π
    if *checkSat*($\pi_{\forall,\int}$) *is Sat by dReal* **then**
      return *counterexample(result.satAssignment)*;
    $\overline{\Psi}^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}} \leftarrow \overline{\Psi}^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}} \wedge \neg\pi_{\forall,\int}$
return True;

A naive implementation of the two-phase solving algorithm may generate many redundant scenarios for hybrid automata. Consider three jump conditions with guards $x > 60$, $y > 10$, and $z > 20$. There are 7 possible scenarios from these guards for a single jump. However, when one jump, say $x > 60$, is taken, the guards for the other jumps, i.e., $y > 10$ and $z > 20$, are not important. It suffices to consider only 3 scenarios, namely, $x > 60$, $y > 10$, and $z > 20$.

We implement a simple algorithm to minimize scenarios. A conjunction of literals π is a scenario if π implies the encoding Ψ. A scenario $\pi = l_1 \wedge \cdots \wedge l_m$ is minimal if $(\neg l_i \wedge \bigwedge_{j \neq i} l_j) \wedge \Psi$, where a literal in π is false, is unsatisfiable. To minimize a scenario π, we apply a dual propagation approach [18]. Because π implies Ψ, the formula $\pi \wedge \neg\Psi$ is unsatisfiable. We evaluate the unsatisfiable core of $\pi \wedge \neg\Psi$ using Z3 to extract a minimal scenario from π.

4.4 Implementation

Figure 4 shows the overall architecture of STLMC. The given STL formula φ is first translated into the ϵ-weakening of the negated formula $(\neg\varphi)^{-\epsilon}$. The SMT encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ for $1 \leq k \leq N$ is then built using the STL bounded model checking algorithm [4,16]. The satisfiability of $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ can be checked directly using an SMT solver or using the two-phase optimization algorithm. Our tool is implemented in around 9,500 lines of Python code.

Simplifying Universal Quantification. The encoding $\Psi^{k,\tau}_{H,(\neg\varphi)^{-\epsilon}}$ includes universal quantification over time. Such $\exists\forall$-conditions are supported by several solvers but involve high computational costs. STLMC implements the following methods to simplify universal quantification. For polynomial hybrid automata, we use the quantifier-free encoding [7] to encode $\exists\forall$-conditions as quantifier-free formulas. For invariant STL properties of the form $p \to \Box_I q$, we reduce redundant universal quantifiers by reducing the model checking problem into the reachability [16].

Parallelizing Two-Phase Solving. We parallelize the two-phase solving algorithm for ODEs. As mentioned, the first phase generates abstract scenarios, and the second phase checks the feasibility of the scenarios. We simply run the feasibility checking of different scenarios in parallel. If any of such scenario is satisfied, then a counterexample is found and all other jobs can be terminated. If all scenarios, checking in parallel, are unsatisfiable, then there is no counterexample. As shown in Sec 5, it greatly improves the performance for the ODE cases in practice.

Supporting Various SMT Solvers. We implement a generic wrapper interface based on the SMT2LIB standard [5] to support various SMT solvers. Therefore, it is easy to extend our tool with a new SMT solvers, if it follows SMT2LIB. Given an input model, STLMC can detect the most suitable solvers. If the model has ODE dynamics and Z3 is selected, the tool raises an exception. Also, when the model has linear dynamics and the dReal is chosen, STLMC gives a warning to use Z3 or Yices instead of dReal.

5 Experimental Evaluation

We evaluate the performance of STLMC to solve robust STL model checking by addressing the following research questions:

RQ1 How effective is STLMC solving robust STL model checking of hybrid automata with various dynamics, including linear, nonlinear and nonlinear ODEs.
RQ2 How effective is the enumeration algorithm compared to only SMT-based model checking algorithm?

We have run all experiments on Intel Xeon 2.8GHz with 256 GB memory.

5.1 RQ1: Robust STL model checking of Hybrid Automata

To evaluate the performance of STLMC, we have performed robust STL model checking considering four versions of our algorithms: one without path enumerations in Sec. 4.2, one with path enumerations in Sec. 4.3, one with minimized path enumerations in Sec. **??**, and with minimized path enumerations using multithread. We consider the following hybrid automata models that are explained in Sec. B: load management of batteries (Bat) and water tank systems (Water) for linear dynamics, autonomous driving car (Car) and railroad gate controller

Table 2: Robust Bounded Model Checking of STL

Dyn.	Model	STL formula	ϵ	$\|Enc\|$	Time	Result	k	Alg.
Linear (N = 20)	Bat	$f_1 : (\Diamond_{[4,10]} p_1)\, \Box_{[4,10]}\, p_2$	0.1	12.9	137	T	20	SMT
		$f_2 : (\Diamond_{[1,5]} p_1)\, \mathbf{R}_{[5,20]}\, p_2$	3.5	2.76	5.71	F	5	SMT
		$f_3 : \Box_{[4,14]}(p_1 \rightarrow \Diamond_{[0,10]} p_2)$	0.1	3.8	22.1	F	8	SMT
	Water	$f_1 : \Box_{[1,3]}(p_1\, \mathbf{R}_{[1,10]}\, p_2)$	2.5	18.8	26.2	T	20	SMT
		$f_2 : (\Diamond_{[1,10]} p_1)\, \mathbf{U}_{[2,5]}\, p_2$	0.1	1.9	4.22	F	4	SMT
		$f_3 : \Diamond_{[4,10]}(p_1 \rightarrow \Box_{[2,5]} p_2)$	0.01	11.2	20.2	T	20	SMT
Poly (N = 10)	Car	$f_1 : \Box_{[0,4]}(p_1 \rightarrow \Diamond_{[2,5]} p_2)$	0.5	2.2	7.24	F	5	SMT
		$f_2 : (\Diamond_{[0,4]} p_1)\, \mathbf{U}_{[0,5]}\, p_2$	2	1.7	6.27	F	3	SMT
		$f_3 : \Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,5]}\, p_2)$	0.1	7.3	9.72	T	10	SMT
	Rail	$f_1 : \Diamond_{[0,5]}(p_1\, \mathbf{U}_{[1,8]}\, p_2)$	1	2.3	3.43	F	5	SMT
		$f_2 : \Diamond_{[0,4]}(p_1 \rightarrow \Box_{[2,10]} p_2)$	5	3.8	0.86	T	10	SMT
		$f_3 : (\Box_{[0,5)} p_1)\, \mathbf{U}_{[2,10]}\, p_2$	4	1.9	2.83	F	4	SMT
ODE (N = 5)	Therm	$f_1 : \Box_{[2,4]}(p_1 \rightarrow \Diamond_{[3,10]} p_2)$	2	0.9	-	TO	2	SMT
					83.3	F	3	Enum
		$f_2 : \Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,\text{inf})}\, p_2)$	0.01	1.2	-	TO	2	SMT
					817	T	5	Enum
		$f_3 : \Box_{[0,10]}(p_1\, \mathbf{R}_{[0,\text{inf})}\, p_2)$	2	1.2	-	TO	2	SMT
					59.3	F	4	Enum
	Oscil	$f_1 : \Diamond_{[0,3]}(p_1\, \mathbf{R}_{[0,\text{inf})}\, p_2)$	0.1	1.5	-	TO	2	SMT
					110	T	5	Enum
		$f_2 : \Diamond_{[2,5]}(\Box_{[0,3]} p_1)$	1	1.2	-	TO	2	SMT
					89.8	F	3	Enum
		$f_3 : (\Box_{[1,3]} p_1)\, \mathbf{R}_{[2,5]}\, p_2$	0.1	1.2	-	TO	2	SMT
					736	F	2	Enum

(Rail) for polynomial dynamics, networked thermostat controllers (Therm) and oscillator (Oscil) for ODEs. For each model, we consider two STL formulas that contain two nested temporal operators.

We have performed robust STL model checking model checking of these STL properties, up to bound $N = 20$ for linear models, $N = 10$ for polynomial models, and $N = 5$ for ODEs models. We assign different time bounds τ_{max} and thresholds ϵ to different models, since they depend on model parameters. As an underlying SMT solver, we use Yices2 [10] for hybrid automata with linear and polynomial dynamics. For hybrid automata with nonlinear dynamics, we use Z3 [8] as an underlying solve to get possible scenarios and use dReal3 [13] to check whether the possible scenario is an actual path of the model.

The experimental results are summarized in Table 2. $|Enc|$ denotes encoding sizes (in thousands), given by the number of connectives in the encoding. The execution times are written in seconds. The result of the model checking is written in *Result* as **T** and **F**. **T** means that the model satisfies the given STL

formula up to bounds and **F** means the model has a counterexample of the formula. For the cases with counterexamples, the table the bound k for which a counterexample is found. For linear and polynomial models, the performance of the smt-based algorithm in 4.2 was the best, so we wrote the results using the smt-based algorithm. We use data obtained using the parallel two-phased algorithm for nonlinear-ODEs.

As shown in Table 2, our tool can deal with nontrivial STL formulas of various models. For ODE models, the solving time is relatively slow compared to linear and polynomial models, due to the complexity of dynamics. Even in that case, all problems could be solved within 15 minutes. There are some cases that the analysis of the large bound yields results in less time than the analysis of the small bound. E.g., for Oscillator, the analysis of the formula f_3 took 736 seconds for $k = 2$, but took 110 seconds for k = 5. This result is due to **Jia: Why??**

5.2 RQ2: Comparison between Enumeration Algorithm and only SMT-based model checking

To evaluate the effectiveness of the enumeration algorithm, we have performed robust STL model checking of hybrid automata with ODE models. We consider two versions of our algorithm: the smt-based algorithm (SMT) and the two-phase algorithm (Enum+). For each model, two STL formulas are considered, one case **T** and one case **F** of the STL formulas used in Section 5.1. We measure the execution time in seconds up to bound 4 with a timeout of 5 hours.

The experimental results are summarized in Table 3. **ID** denotes the formula label in Table 2. The execution times are written in seconds. The result of the model checking is written in *Result* For the cases with counterexamples, the table the bound n for which a counterexample is found.

As shown in Table 3, Enum⋆significantly outperforms SMTfor nonlinear-ODEs models. In particular, in case of SMT, More than half of experiments are timeouts while all problems can be solved in time using Enum⋆. E.g, for Card, the analysis of φ_1 took only $1,543$ seconds in Enum⋆, but the same problem is timeout using SMT.

Table 3: Analyzing SMTand Enum⋆algorithms

Model	Alg.	φ_1				φ_2			
		ID	Time	Result	k	ID	Time	Result	k
Therm	Enum	f_1	173	T	4	f_3	179	F	4
	SMT		11.2	T	3		1544	F	2
Oscil	Enum	f_1	29.89	T	4	f_2	372	F	3
	SMT		11.2	T	3		1544	F	2

6 Related Work

...

7 Concluding Remarks

...

References

1. Alur, R.: Principles of cyber-physical systems. The MIT Press (2015)
2. Angeli, D., Sontag, E., Wang, Y.: A characterization of integral input-to-state stability. IEEE Transactions on Automatic Control **45**(6), 1082–1097 (2000). https://doi.org/10.1109/9.863594
3. Bae, K., Gao, S.: Modular smt-based analysis of nonlinear hybrid systems. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 180–187. IEEE (2017)
4. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3**(POPL) (jan 2019). https://doi.org/10.1145/3290364, https://doi.org/10.1145/3290364
5. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England). vol. 13, p. 14 (2010)
6. Chan, N., Mitra, S.: Verifying safety of an autonomous spacecraft rendezvous mission (03 2017)
7. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free smt encoding of non-linear hybrid automata. In: Formal Methods in Computer-Aided Design (FMCAD), 2012. pp. 187–195. IEEE (2012)
8. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. p. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
9. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for stl. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 264–279. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification. pp. 737–744. Springer International Publishing, Cham (2014)
11. Fox, M., Long, D., Magazzeni, D.: Plan-based policies for efficient multiple battery load management. Journal of Artificial Intelligence Research **44**, 335–382 (2012)
12. Gao, S., Avigad, J., Clarke, E.M.: δ-. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning. pp. 286–300. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
13. Gao, S., Kong, S., Clarke, E.M.: dreal: An smt solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) Automated Deduction – CADE-24. pp. 208–214. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
14. Henzinger, T.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems, NATO ASI Series, vol. 170, pp. 265–292. Springer (2000)

15. Huang, Z., Fan, C., Mereacre, A., Mitra, S., Kwiatkowska, M.: Invariant verification of nonlinear hybrid automata networks of cardiac cells. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification. pp. 373–390. Springer International Publishing, Cham (2014)
16. Jia Lee, Geunyeol Yu, K.B.: Efficient smt-based model checking for signal temporal logic. In: IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE (2021)
17. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. Lecture Notes in Computer Science, vol. 3253, pp. 152–166. Springer (2004)
18. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: 2014 Formal Methods in Computer-Aided Design (FMCAD). pp. 179–186. IEEE (2014)
19. Raisch, J., Klein, E., Meder, C., Itigin, A., O'Young, S.: Approximating automata and discrete control for continuous systems — two examples from process control. In: Hybrid systems V. pp. 279–303. Springer (1999)
20. Sebastiani, R.: Lazy satisfiability modulo theories. Journal on Satisfiability, Boolean Modeling and Computation **3**(3-4), 141–224 (2007)

A The Proofs of Lemmas

Lemma 1. *For a signal σ and an STL property φ:*

(1) $\rho(\varphi,\sigma,t) > \epsilon \;\Rightarrow\; \sigma,t \models_\tau \varphi^{+\epsilon}$ *(3)* $\sigma,t \models_\tau \varphi^{+\epsilon} \;\Rightarrow\; \rho(\varphi,\sigma,t) \geq \epsilon$
(2) $\sigma,t \models_\tau \varphi^{-\epsilon} \;\Rightarrow\; \rho(\varphi,\sigma,t) \geq -\epsilon$ *(4)* $\rho(\varphi,\sigma,t) > -\epsilon \;\Rightarrow\; \sigma,t \models_\tau \varphi^{-\epsilon}$

Proof. (1). The proof is by structural induction over the formula φ

Suppose $\varphi = x \geq 0$ and $\rho(x \geq 0,\sigma,t) > \epsilon$. By definition, $x(\sigma(t)) > \epsilon$ iff $\rho(x - \epsilon \geq 0,\sigma,t) > 0$. By Theorem **??**, if $\rho(x - \epsilon \geq 0,\sigma,t) > 0$, then $\sigma,t \vDash x - \epsilon \geq 0$. Thus, $\sigma,t \vDash \varphi^{+\epsilon}$. Suppose $\varphi = x > 0$ and $\rho(x \geq 0,\sigma,t) > \epsilon$. $\sigma(x(t)) > \epsilon$ by assumption. The equation can be rewritten to $\sigma,t \vDash x > \epsilon$ iff $\sigma,t \vDash \varphi^{+\epsilon}$. Suppose $\varphi = \neg\phi$ and $\rho(\neg\phi,\sigma,t) > \epsilon$. By definition, $-\rho(\phi,\sigma,t) > \epsilon$ iff $\rho(\phi,\sigma,t) < -\epsilon$. By induction hypothesis, $\sigma,t \nvDash \phi^{-\epsilon}$. Thus, $\sigma,t \vDash \varphi^{+\epsilon}$, since $\neg(\phi^{-\epsilon}) \equiv (\neg\phi)^{+\epsilon}$. Suppose $\varphi = \varphi_1 \wedge \varphi_2$ and $\rho(\varphi,\sigma,t) > \epsilon$. By definition, $\rho(\varphi_1,\sigma,t) > \epsilon$ and $\rho(\varphi_2,\sigma,t) > \epsilon$. By induction hypothesis, $\sigma,t \vDash \varphi_1^{+\epsilon}$ and $\sigma,t \vDash \varphi_2^{+\epsilon}$. Thus, $\sigma,t \vDash \varphi^{+\epsilon}$, Suppose $\varphi = \varphi_1 \,\mathbf{U}_I\, \varphi_2$ and $\rho(\varphi,\sigma,t) > \epsilon$. By definition, $\sup\{\min(\rho(\varphi_2,\sigma,t'),\ \text{least}(\varphi_1,\sigma,t,t')) \,|\, t' \in t + I\} > \epsilon$, where $\text{least}(\varphi_1,\sigma,t,t')) = \inf\{\rho(\varphi,\sigma,t'') \,|\, t'' \in [t,t']\}$. It means, $\exists t' \in t+I, \rho(\varphi_2,\sigma,t') > \epsilon$ and $\inf\{\rho(\varphi,\sigma,t'') \,|\, t'' \in [t,t']\} > \epsilon$. If $\inf\{\rho(\varphi,\sigma,t'') \,|\, t'' \in [t,t']\} > \epsilon$, then $\forall t'' \in [t,t']$, $\rho(\varphi_1,\sigma,t'') > \epsilon$. By induction hypothesis, $\exists t' \in t+I$, $\sigma,t' \vDash \varphi_2^{+\epsilon}$ and $\forall t'' \in [t,t']$, $\sigma,t'' \vDash \varphi_1^{+\epsilon}$. Thus, $\sigma,t \vDash (\varphi_1 \,\mathbf{U}\, \varphi_2)^{+\epsilon}$

Suppose $\sigma,t \vDash x > -\epsilon$. Then, $\sigma(x(t)) > -\epsilon$ and $\sigma(x(t)) = \rho_\tau(x > 0,\sigma,t)$. Thus, $\rho_\tau(x > 0,\sigma,t) > -\epsilon \geq -\epsilon$ Suppose $\sigma,t \vDash x \geq -\epsilon$. Then, $\sigma(x(t)) \geq -\epsilon$ and $\sigma(x(t)) = \rho_\tau(x > 0,\sigma,t)$. Thus, $\rho_\tau(x > 0,\sigma,t) \geq -\epsilon$ Suppose, $\sigma,t \vDash (\neg\varphi^{-\epsilon})$. Then, $\sigma,t \nvDash \varphi^{+\epsilon}$ by definition. Then $\rho_\tau(\varphi,\sigma,t) \leq \epsilon$ by induction hypothesis. Thus, $-\rho_\tau(\varphi,\sigma,t) \geq -\epsilon$ iff $\rho_\tau(\neg\varphi,\sigma t) \geq -\epsilon$. Suppose, $\sigma,t \vDash (\varphi_1 \wedge \varphi_2)^{-\epsilon}$. Then,

$\sigma, t \vDash \varphi_1^{-\epsilon} \wedge \sigma, t \vDash \varphi_2 - \epsilon$ By induction hypothesis, $\rho_\tau(\varphi_1, \sigma, t) \wedge \rho_\tau(\varphi_2, \sigma, t) \geq -\epsilon$. Thus, $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t))) \geq -\epsilon$ Suppose, $\sigma, t \vDash \varphi^{-\epsilon}$. It means, $\sigma, t \vDash \varphi_1^{-\epsilon}\ \mathbf{U}_I\ \varphi_2^{-\epsilon}$ by definition. Then, $\exists t' \in t + I, \sigma, t' \vDash \varphi_2^{-\epsilon}$, $\forall t'' \in [t, t'], \sigma, t \vDash \varphi_1^{-\epsilon}$. By induction hypothesis, there is a time point $t' \in t + I$ such that $\rho_\tau(\varphi_2, \sigma, t') \geq -\epsilon \wedge \forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$. Thus, $\exists t' \in t + I, \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t'')) \geq -\epsilon$. Therefore, $\rho_\tau(\varphi_1\, \mathbf{U}_I\, \varphi_2, \sigma, t) = \sup_{t' \in t+I} (\min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t''))) \geq -\epsilon$

(2). Suppose $\sigma, t \vDash x > \epsilon$. Then, $\sigma(x(t)) > \epsilon$. Thus, $\rho_\tau(\varphi, \sigma, t) \geq \epsilon$. Suppose $\sigma, t \vDash x > \epsilon$. Then, $\sigma, t \vDash x \geq \epsilon$ iff $\sigma(s(t)) \geq e$ Therefore, $(\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$ Suppose $\sigma, t \vDash (\neg\varphi')^{+\epsilon}$. Then, $sigma, t \nvDash \varphi'^{-\epsilon}$ by definition. And $\rho_\tau(\varphi', \sigma, t) \leq -\epsilon$ by induction hypothesis. Thus, $\rho_\tau(\neg\varphi', \sigma, t) \geq \epsilon$. Suppose $\sigma, t \vDash (\varphi_1 \wedge \varphi_2)^{+\epsilon}$. Then, $\sigma, t \vDash \varphi_1^{+\epsilon} \wedge \sigma, t \vDash \varphi_2^{+\epsilon}$. The above formula implies $\rho_\tau(\varphi_1, \sigma, t) \geq \epsilon \wedge \rho_\tau(\varphi_1, \sigma, t) \geq \epsilon$ by induction hypothesis. Then, $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_1, \sigma, t))$ is also greater than or equal to ϵ. Suppose $\sigma, t \vDash (\varphi_1\, \mathbf{U}_I\, \varphi_2)^{+\epsilon}$. Then, $\exists t' \in t + I, \sigma, t' \vDash \varphi_2^{+\epsilon}, \forall t'' \in [t, t'], \sigma, t'' \vDash \varphi_1 + \epsilon$. If then, $\exists t' \in t + I, \rho_\tau(\varphi_2, \sigma, t') \geq \epsilon \wedge \forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t) \geq \epsilon$. Thus, $\exists t' \in t + I, \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t)) \geq \epsilon$. The above formula implies $\rho_\tau(\varphi_1\, \mathbf{U}_I\, \varphi_2, \sigma, t) \geq \epsilon$

Suppose $\rho_\tau(x > 0, \sigma, t) > -\epsilon$. Then, $\sigma(x(t)) > -\epsilon$ by definition. The formula implies $\sigma, t \vDash x > -\epsilon$. Suppose $\rho_\tau(x \geq 0, \sigma, t) > -\epsilon$. Then, $\sigma(x(t)) > -\epsilon$ by definition. The formula implies $\sigma, t \vDash x > -\epsilon$. If then, $\sigma, t \vDash x \geq -\epsilon$ Suppose $\rho_\tau(\neg\varphi', \sigma, t) > -\epsilon$. Then, $\rho_\tau(\varphi', \sigma, t) < \epsilon$ by definition. The formula implies $\sigma, t \nvDash \varphi'^{+\epsilon}$ by induction hypothesis. The formula is equivalent to $\sigma, t \vDash (\neg\varphi')^{-\epsilon}$ by definition. Suppose $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) > -\epsilon$. Then, $\rho_\tau(\varphi_1, \sigma, t) > -\epsilon \wedge \rho_\tau(\varphi_2, \sigma, t) > -\epsilon$. By induction hypothesis, $\sigma, t \vDash \varphi_1^{-\epsilon} \wedge \sigma, t \vDash \varphi_2^{-\epsilon}$. The formula implies $\sigma t \vDash (\varphi_1 \wedge \varphi_2)^{-\epsilon}$ Suppose $\rho_\tau(\varphi_1\, \mathbf{U}_I\, \varphi_2, \sigma, t) > -\epsilon$. Then, $\exists t' \in t + I, \rho_\tau(\varphi_2, \sigma, t') > -\epsilon \wedge \forall t'' \in [t, t'] \rho_\tau(\varphi_1, \sigma, t'') > -\epsilon$. The formula implies $\exists t' int + I, \sigma, t' \vDash \varphi_2 \wedge \forall t'' \in [t, t'], \sigma, t'' \vDash \varphi_1$ by induction hypothesis. Therefore, $\sigma, t \vDash (\varphi_1\, \mathbf{U}_I\, \varphi_2)^{-\epsilon}$

Theorem 2. *For a hybrid automaton H, an STL formula φ, a time t, a time bound τ, and an arbitrary positive real value ϵ, the following properties are hold:*

(1) $\exists \sigma \in H.\ \sigma, t \models_\tau (\neg\varphi)^{-\epsilon} \implies \exists \sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$
(2) $\forall \sigma \in H,\ \sigma, t \not\models_\tau (\neg\varphi)^{-\epsilon} \implies \forall \sigma \in H,\ \rho_\tau(\varphi, \sigma, t) \geq \epsilon$

Proof. **To be done!**

K: The proof must be given

B Case Study

In this section, we explain various hybrid automata that we used in Section 5.

B.1 Load Management of Batteries

There are two fully charged batteries, and a control system switches load between these batteries (adapted from [19]). The total charge g_i and kinetic energy d_i of

battery $i = 1, 2$, changes according to the ODEs in three modes *On* (O), *Off* (F) and *Dead* (D). A controller goes to the *Dead* mode if the total charge g_i is less then or equal to the given threshold $c > 0$. Figure 5 shows a hybrid automaton H_1 of one battery component. (the other component is similar).

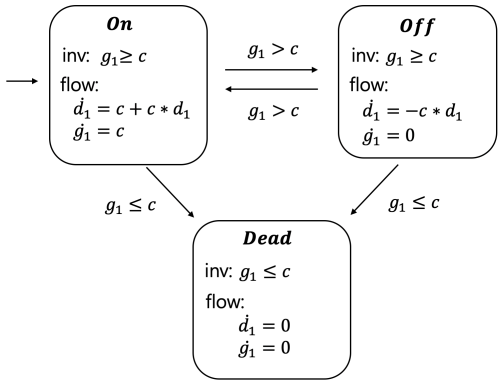


Fig. 5: A hybrid automaton of one battery controller H_1

For the experiments, we consider the following approximate dynamics to obtain linear flow conditions.

(O_1O_2) $\dot{d}_1 = 1$, $\dot{g}_1 = -0.3$, $\dot{d}_2 = 1$, $\dot{g}_2 = -0.3$

(O_1F_2) $\dot{d}_1 = 0.8$, $\dot{g}_1 = -0.5$, $\dot{d}_2 = -0.166$, $\dot{g}_2 = 0$

(F_1O_2) $\dot{d}_1 = -0.166$, $\dot{g}_1 = 0$, $\dot{d}_2 = 0.8$, $\dot{g}_2 = -0.5$

(O_1D_2) $\dot{d}_1 = 0.7$, $\dot{g}_1 = -7$, $\dot{d}_2 = 0$, $\dot{g}_2 = 0$

(D_1O_2) $\dot{d}_1 = 0$, $\dot{g}_1 = 0$, $\dot{d}_2 = 0.7$, $\dot{g}_2 = -7$

(D_1D_2) $\dot{d}_1 = 0$, $\dot{g}_1 = 0$, $\dot{d}_2 = 0$, $\dot{g}_2 = 0$

Table 4 shows the STL formulas (for RQ1) used in the experiments.

Figure 6 is a model file of the load management of two batteries in STLMC.

B.2 Water Tank Systems

There are two connected water tanks, where water flows from one tank to another (adapted from [11]). The water level x_i of tank $i = 1, 2$ changes according to the ODEs in two modes *On* and *Off*, where a and q_i depend on the the pump's power and the width of the pipe, and g is the standard gravity constant. Figure 7 shows a hybrid automaton H_1 of one water tank (the other component is similar).

For the experiments, we consider the following dynamics to obtain linear flow conditions.

```
real b1; real b2;
[0, 10] g1; [0, 10] g2;
[0, 10] d2; [0, 10] d1;
# state
# 1: ON, 2: OFF, 3: DEAD
{
  mode: b1 = 1; b2 = 1;
  inv:  g1 >= 0.5; g2 >= 0.5;
  flow:
    d/dt[d1] = 1; d/dt[d2] = 1;
    d/dt[g1] = -0.3; d/dt[g2] = -0.3;
  jump:
    g1 >= 2 =>
        (and (b1' = 1) (b2' = 2) (d1' = d1)
          (d2' = d2) (g1' = g1) (g2' = g2));
    g1 <= 1 =>
        (and (b1' = 3) (b2' = 1) (d1' = d1)
          (d2' = d2) (g1' = g1) (g2' = g2));
    g2 >= 2 =>
        (and (b1' = 2) (b2' = 1) (d1' = d1)
          (d2' = d2) (g1' = g1) (g2' = g2));
    g2 <= 1 =>
        (and (b1' = 1) (b2' = 3) (d1' = d1)
          (d2' = d2) (g1' = g1) (g2' = g2));

}
{
  mode: b1 = 1; b2 = 2;
  inv:  g1 >= 0.5;
  flow:
    d/dt[d1] = 0.8; d/dt[d2] = -0.166;
    d/dt[g1] = -0.5; d/dt[g2] = 0;
  jump:
    g2 >= 2 =>
       (and (b1' = 2) (b2' = 1) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));
    g1 <=1 =>
       (and (b1' = 3) (b2' = 1) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));

}
{
  mode: b1 = 2; b2 = 1;
  inv:  g2 >= 0.5;
  flow:
    d/dt[d1] = -0.166; d/dt[d2] = 0.8;
    d/dt[g1] = 0; d/dt[g2] = -0.5;
  jump:
    g1 >= 2 =>
       (and (b1' = 1) (b2' = 2) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));
    g2 <= 1 =>
       (and (b1' = 1) (b2' = 3) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));

}
{
  mode: b1 = 3; b2 = 1;
  inv:  g1 <= 1; g2 >= 0.5;
  flow:
    d/dt[d1] = 0; d/dt[d2] = 0.7;
    d/dt[g1] = 0; d/dt[g2] = -7;
  jump:
    g2 <= 1 =>
       (and (b1' = 3) (b2' = 3) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));

}
{
  mode: b1 = 1; b2 = 3;
  inv:  g1 >= 0.5; g2 <= 1;
  flow:
    d/dt[d1] = 0.7; d/dt[d2] = 0;
    d/dt[g1] = -7; d/dt[g2] = 0;
  jump:
    g1 <= 1 =>
       (and (b1' = 3) (b2' = 3) (d1' = d1)
         (d2' = d2) (g1' = g1) (g2' = g2));

}
{
  mode: b1 = 3; b2 = 3;
  inv:
  flow:
    d/dt[d1] = 0; d/dt[d2] = 0;
    d/dt[g1] = 0; d/dt[g2] = 0;
  jump:
}

init:
(and (b1 = 1) (b2 = 1) (8.4 <= g1) (g1 <= 8.5)
  (0 <= d1) (d1 <= 0.1) (7.4 <= g2) (g2 <= 7.5)
  (0 <= d2) (d2 <= 0.1)) ;

propositions:

#timebound 30
goal:
@f1: (<>[1, 5] g2 <= 5) R[5, 20] (d2 >= 4.5);
@f2: <>[10, 15] ((g1 <= 3) U[1, 4] (b1 = 3));
@f3: <>[4, 10] ((d2 <= 4) -> []4, 10] (g2 <= 5));
```

Fig. 6: An input model of the load management of two batteries

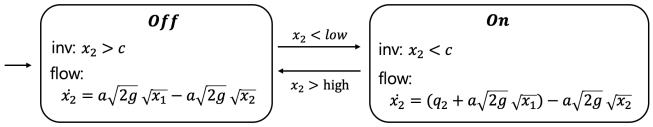


Fig. 7: A hybrid automaton of one water tank controller H_1

Table 4: STL for robust STL model checking of battery

ID	STL formula	Explanation
φ_1:	$(\Diamond_{[1,5]}(g_2 \leq 5))\, \mathbf{R}_{[5,20]}(d_2 \geq 4.5)$	For time interval $[5, 20]$, d_2 is greater than or equal to 4.5, until g_2 is less than equal to 5 with in time interval $[1, 5]$.
φ_2:	$\Diamond_{[10,15]}((g_1 \leq 3)\, \mathbf{U}_{[1,4]}\, \text{Off}_2)$	Within time interval $[10, 15]$, g_2 is less than or equal to 3, until battery_2 is off.
φ_3:	$(\Box_{[1,5]}(g_1 > 1))\ \mathbf{U}_{[2,8.5)}\ (d_1 \geq 2)$	g_1 is greater than 1 for 4 time units, until d_1 is greater than or equal to 2

$$\dot{x}_1 = \begin{cases} -0.8 & \text{(On)} \\ 0.9 & \text{(Off)} \end{cases} \qquad \dot{x}_2 = \begin{cases} -0.6 & \text{(On)} \\ 1 & \text{(Off)} \end{cases}$$

Table 5 shows the STL formulas (for RQ1) used in the experiments.

Table 5: STL for robust STL model checking of water tank system

ID	STL formula	Explanation
φ_1:	$(\Diamond_{[1,5]}(x_2 < 5.5))\, \mathbf{U}_{[2,5]}(x_1 \geq 5)$	Within time interval $[1, 5]$, x_2 is less than 5.5, until x_1 is greater than or equal to 5 with in time interval $[2, 5]$.
φ_2:	$\Box_{[1,3]}((x_1 \leq 7)\, \mathbf{R}_{[1,10]}(x_2 \leq 3))$	For time interval $[1, 3]$, x_2 is greater than or equal to 3, until x_1 is less than or equal to 7 for time interval $[1, 3]$.
φ_3:	$\Box_{[2,8]}(\Diamond_{[4,10]}(x_1 \geq 5.42))$	For time interval $[2, 8]$, x_1 is greater than or equal to 5.42 within time interval $[4, 10]$.

Figure 8 is a model file of the water tank systems in STLmc.

B.3 Autonomous Car

We consider two cars: one drives according to its own scenario, and the other follows the leader. We consider a polynomial dynamics case (adapted from [1]). The hybrid automaton for the polynomial case is shown in Fig. 9. In this model, we do not explicit specify the positions of the cars. The positions of the cars are represented using the *relative* distance (r_x, r_y), and the follower's velocity

```
bool a; bool b;
[0, 10] x1; [0, 10] x2;
{
  mode: a = false; b = false;
  inv:  x1 >= 1; x2 >= 1;
  flow:
    d/dt[x1] = -0.8; d/dt[x2] = -0.6;
  jump:
    x1 <= 2 =>
       (and (a') (not b')
         (x1' = x1) (x2' = x2));
    x2 <= 3 =>
       (and (not a') b'
         (x1' = x1) (x2' = x2));
    (or (x1 <= 2) (x2 <= 3)) =>
          (and (a') (b')
            (x1' = x1) (x2' = x2));
}
{
  mode: a = false; b = true;
  inv:  x1 >= 1; x2 <= 9;
  flow:
    d/dt[x1] = -0.8; d/dt[x2] = 1;
  jump:
    x2 >= 6 =>
       (and (not a') (not b')
         (x1' = x1) (x2' = x2));
    x1 <= 2 =>
       (and a' b' (x1' = x1) (x2' = x2));
    (or (x2 >= 6) (x1 <= 2)) =>
          (and a' (not b')
           (x1' = x1) (x2' = x2));
}
{
  mode: a = true; b = false;
  inv:  x1 <= 8; x2 >= 1;
  flow:
    d/dt[x1] = 0.9; d/dt[x2] = -0.6;
  jump:
    x1 >= 5 =>
       (and (not a') (not b')
         (x1' = x1) (x2' = x2));
    x2 <= 3 =>
        (and a' b' (x1' = x1) (x2' = x2));
    (or (x1 >=5) (x2 <= 3)) =>
          (and (not a') (b')
           (x1' = x1) (x2' = x2));
}
{
  mode: a = true; b = true;
  inv:  x1 <= 9; x2 <= 9;
  flow:
    d/dt[x1] = 0.9; d/dt[x2] = 1;
  jump:
    x1 >= 5 =>
       (and (not a') b'
         (x1' = x1) (x2' = x2));
    x2 >= 6 =>
       (and a' (not b')
         (x1' = x1) (x2' = x2));
    (or (x1 >= 5) (x2 >=6)) =>
          (and (not a') (not b')
            (x1' = x1) (x2' = x2));
}
init:
(and (not(a)) not(b) (4.4 <= x1) (x1 <= 4.5)
   (5.9 <= x2) (x2 <= 6));

propositions:
pb = b;

# timebound : 20
goal:
@f1: (<>[1, 10) x2 < 5.5) U[2, 5] (x1 >= 5);
@f2: [][1, 3]((x1 <= 7) R[1, 10] (x2 <= 3));
@f3: [][2, 8] (<>[4, 10] x1 >= 5.42);
@f4: <>[4, 10] (x1 >= 4 -> [][2, 5] x2 <= 2);
```

Fig. 8: An input model of the water tank systems

(v_{x_1}, v_{y_1}) changes according to the ODEs based on the relative distance as follows:

$$\begin{aligned}
&\dot{r}_x = v_{x_1}, \quad \dot{r}_y = v_{y_1}, \quad \dot{v}_{x_1} = 1.2, \quad \dot{v}_{y_1} = 1.4 && (C_xC_y)\\
&\dot{r}_x = v_{x_1}, \quad \dot{r}_y = v_{y_1}, \quad \dot{v}_{x_1} = 1.2, \quad \dot{v}_{y_1} = -1.4 && (C_xF_y)\\
&\dot{r}_x = v_{x_1}, \quad \dot{r}_y = v_{y_1}, \quad \dot{v}_{x_1} = -1.2, \quad \dot{v}_{y_1} = 1.4 && (F_xC_y)\\
&\dot{r}_x = v_{x_1}, \quad \dot{r}_y = v_{y_1}, \quad \dot{v}_{x_1} = -1.2, \quad \dot{v}_{y_1} = -1.4 && (F_xF_y)
\end{aligned}$$

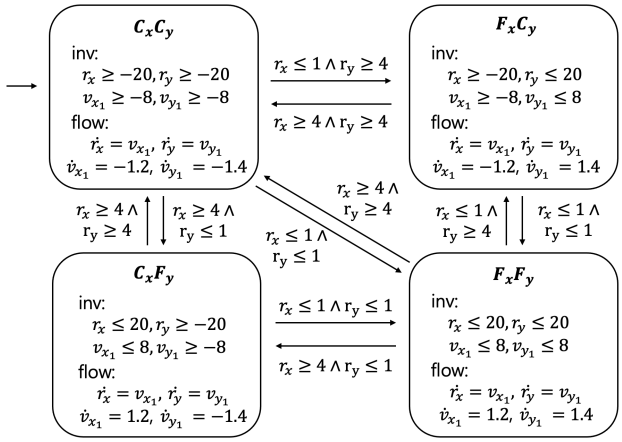


Fig. 9: The hybrid automaton for autonomous car (F: Far, C: Close)

Table 6 shows the STL formulas (for RQ1) used in the experiments.

Table 6: STL for robust STL model checking of autonomous car

ID	STL formula	Explanation
φ_2:	$\Box_{[5,10]}(\Diamond_{[4,10]}(r_y \geq 4))$	For time interval $[5, 10]$, r_y is greater than or equal to 4 within time interval $[4, 10]$.

Figure 10 is a model file of the autonomous car in STLMC.

B.4 Railroad Gate Controller

There is a crossing bar on a circular railroad track and a train runs on the track. Figure 11 shows a hybrid automaton for this system (adapted from [14]). The

```
bool ix; bool iy;
[-40, 40] rx; [-40, 40] ry;
[-30, 30] vx1; [-30, 30] vy1;

#  car1 : inc, inc
{
  mode: ix = true; iy = true;
  inv:  rx < 20; ry < 20;
        vx1 < 8; vy1 < 8;
  flow:
    d/dt[rx] = vx1; d/dt[ry] = vy1;
    d/dt[vx1] = 1.2; d/dt[vy1] = 1.4;
  jump:
   (and (rx >= 3) (ry < 3)) =>
       (and (ix' = false) (iy' = true)
        (rx' = rx) (ry' = ry)
        (vx1' = 0) (vy1' = 0));
   (and (rx < 3) (ry >= 3)) =>
       (and (ix' = true) (iy' = false)
        (rx' = rx) (ry' = ry)
        (vx1' = 0) (vy1' = 0));
}
#  car1 : inc, dec
{
  mode: ix = true; iy = false;
  inv: rx < 20;  ry > -20;
       vx1 < 8;  vy1 > -8;
  flow:
    d/dt[rx] = vx1; d/dt[ry] = vy1;
    d/dt[vx1] = 1.2; d/dt[vy1] = -1.4;
  jump:
    (and (rx >= 3) (ry >= 3)) =>
        (and (ix' = false) (iy' = false)
         (rx' = rx) (ry' = ry)
         (vx1' = 0) (vy1' = 0));
    and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
          (rx' = rx) (ry' = ry)
          (vx1' = 0) (vy1' = 0));
}
#  car1 : dec, inc
{
  mode: ix = false; iy = true;
  inv: rx > -20; ry < 20;
       vx1 > -8; vy1 < 8;
  flow:
    d/dt[rx] = vx1; d/dt[ry] = vy1;
    d/dt[vx1] = -1.2; d/dt[vy1] = 1.4;
  jump:
    (and (rx < 3) (ry < 3)) =>
          (and (ix' = true) (iy' = true)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
    (and (rx >= 3) (ry >= 3)) =>
          (and (ix' = false) (iy' = false)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
    (and (rx < 3) (ry >= 3)) =>
          (and (ix' = true) (iy' = false)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
}
#  car1 : dec, dec
{
  mode: ix = false; iy = false;
  inv:  rx > -20; ry > -20;
        vx1 > -8; vy1 > -8;
  flow:
    d/dt[rx] = vx1; d/dt[ry] = vy1;
    d/dt[vx1] = -1.2; d/dt[vy1] = -1.4;
  jump:
    (and (rx < 3) (ry < 3)) =>
          (and (ix' = true) (iy' = true)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
    (and (rx >= 3) (ry < 3)) =>
          (and (ix' = false) (iy' = true)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
    (and (rx < 3) (ry >= 3)) =>
          (and (ix' = true) (iy' = false)
            (rx' = rx) (ry' = ry)
            (vx1' = 0) (vy1' = 0));
}


init:
(and not(ix) not(iy) (0 <= rx) (rx <= 1)
   (0 <= ry) (ry <= 1) (-2.5 <= vx1)
   (vx1 <= -2) (-2.5 <= vy1) (vy1 <= -2));

propositions:
pix = ix;
piy = iy;

# timebound 25
goal:
@f1: [][5,10] ((vx1 < -2) -> <>[4,10] rx <= -2);
@f2: [][5,10] (<>[4,10] ry >= 4);
@f3: (<>[0, 8] ry > 3) U [0, 15] (vy1 > 1.5);
@f4: <>[0,5] (rx < -2 -> [] [0,5] (vx1 > 4));
```

Fig. 10: An input model of the autonomous car

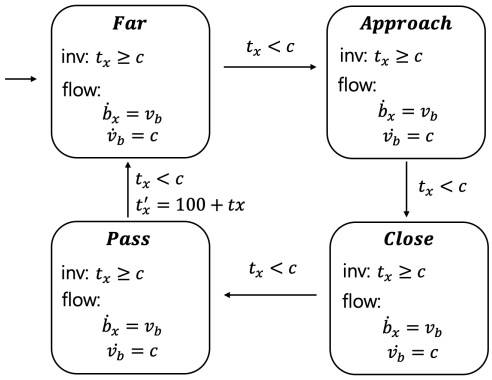


Fig. 11: A hybrid automaton of the railroad

crossing bar is open when the train is closed to the bar. The velocity of the crossing bar v_{b_x} according to the ODEs in four modes *Far*, *Approach*, *Close*, and *Pass*.

For the experiments, we consider the following approximate polynomial dynamics.

$$\begin{aligned}
\dot{t}_x &= -30, & \dot{b}_x &= v_b, & \dot{v}_b &= 0 && (Far)\\
\dot{t}_x &= -5, & \dot{b}_x &= v_b, & \dot{v}_b &= 0.3 && (Approach)\\
\dot{t}_x &= -5, & \dot{b}_x &= v_b, & \dot{v}_b &= 0.5 && (Close)\\
\dot{t}_x &= -5, & \dot{b}_x &= v_b, & \dot{v}_b &= -1 && (Past)
\end{aligned}$$

Table 7 shows the STL formulas (for RQ1) used in the experiments.

Table 7: STL for robust STL model checking of railroad gate controller

ID	STL formula	Explanation
φ_1:	$\square_{[0,5)}((t_x < 50)\,\mathbf{U}_{[2,10]}(b_x > 5))$	For time interval $[0, 5)$, t_x is less than 50, until b_x is greater than 5.
φ_2:	$\Diamond_{[0,4]}((t_x < 50) \rightarrow \square[2, 10](b_x > 40))$	Within 4 time units, if t_x is less than 50, then b_x is greater than 40 for time interval $[2, 10]$.

Figure 12 is a model file of the railroad gate controller in STLMC.

```
bool a; bool b;
[-20, 100] tx; [0, 90] bx;
[-50, 50] vb;
{
  mode: a = false; b = false;
  inv:  tx > 60;
  flow:
    d/dt[tx] = -30; d/dt[bx] = 0;
    d/dt[vb] = 0;
  jump:
    tx < 80 =>
       (and (not a') b' (bx' = bx)
          (tx' = tx) (vb' = 6));
    tx < 70 =>
       (and a' (not b') (bx' = bx)
          (tx' = tx) (vb' = 8));
}
{
  mode: a = false; b = true;
  inv:  tx > 20;
  flow:
    d/dt[tx] = -5; d/dt[bx] = vb;
    d/dt[vb] = 0.3;
  jump:
    tx < 80 =>
       (and a' (not b') (bx' = bx)
          (tx' = tx) (vb' = vb));
}
{
  mode: a = true; b = false;
  inv:  tx > 10;
  flow:
    d/dt[tx] = -5; d/dt[bx] = vb;
    d/dt[vb] = 0.5;
  jump:
    tx < 20 =>
       (and a' b' (bx' = bx)
          (tx' = tx) (vb' = -4));
}
{
  mode: a = true; b = true;
  inv:  tx > -10;
  flow:
    d/dt[tx] = -5; d/dt[bx] = vb;
    d/dt[vb] = -1;
  jump:
    (tx < 0) =>
        (and (not a') (not b')
           (bx' = bx) (vb' = 0)
           (tx' = (100 + tx)));
}
init:
(and (a = false) (b = false) (0<= bx)
     (bx <= 0.5) (89 <= tx) (tx <= 90)
     (0 <= vb) (vb <= 0.1)) ;

propositions:

# timebound 20
goal:
@f1: [][0.0,5.0) (tx < 50) U[2, 10] bx > 5;
@f2: <>[0, 5] ((bx >= 40) U[1, 8] (tx < 40));
@f3: <>[0, 4](tx < 50 -> [][2,10] bx > 40);
```

Fig. 12: An input model of the railroad gate controller

B.5 Spacecraft

There are two spaceships, chaser and target, in $\mathbb{R}^2$. The chaser tries to approach to the target [6]. The *target* spaceship rotates Earth at constant angular velocity ω, and the *chaser* spaceship follows the target. The horizontal and vertical distance, between the target and the chaser are x and y, respectively, which change according to the relative velocities v_x and v_y with thrusts F_x and F_y. The dynamics is as follows:

$$\dot{x} = v_x, \quad \dot{y} = v_y, \quad \dot{v_x} = 3\omega^2 x + 2\omega v_y + \frac{F_x}{m_c}, \quad \dot{v_y} = -2\omega v_x + \frac{F_y}{m_c},$$

where m_c is the mass of the *chaser* spacecraft. The chaser tries to keep the distance between the two spaceship as small as possible.

Figure 13 shows a hybrid automaton of the rendezvous mission. There are three modes (Far, Close, and Recovery) of assigning different values to F_x and F_y. Initially, the (horizontal and vertical) distances x and y are between 60 and 70, and the *chaser* spacecraft starts its mission in the Far mode, moving toward the target. When each distance between the chaser and target is less than 50, the chaser decreases its thrust to keep the velocities v_x and v_y less than -4 to avoid collision, resulting in the Close mode. When the velocity is still too fast, the chaser falls into the to the Recovery mode to change its thrust to the opposite direction.

Table 8 shows the STL formulas (for RQ1 and RQ2) used in the experiments.

Table 8: STL for robust STL model checking of spacecraft

ID	STL formula	Explanation
φ_1	$\square_{[0,5]}\,(((x + y < 70) \wedge (v_x + v_y < -10)) \rightarrow \Diamond_{[0,10)}(v_x + v_y > 4))$	For 5 seconds, if the sum of distances x and y and the sum of velocities v_x and v_y are less than 70 and -10 respectively, the sum of velocities will be greater than 4 after sometime within 10 seconds.
φ_2	$\Diamond_{[0,5]}\,(\square_{[5,9]}\,(\lvert x + y\rvert \leq 50))$	At some time in the first 5 seconds, the absolute value of the sum of distances x and y is less than or equal to 50 for 4 seconds.
φ_3	$\square_{[0,10]}\,((v_x + v_y \leq -8) \rightarrow \Diamond_{[5,15]}\,(v_x + v_y \geq 0))$	For 10 seconds, if the sum of velocities v_x and v_y is less than or equal to -8, the sum of velocities will be greater than equal to 0 after sometime within $[5, 15]$.

Figure 14 is a model file of the spacecraft in STLMC.

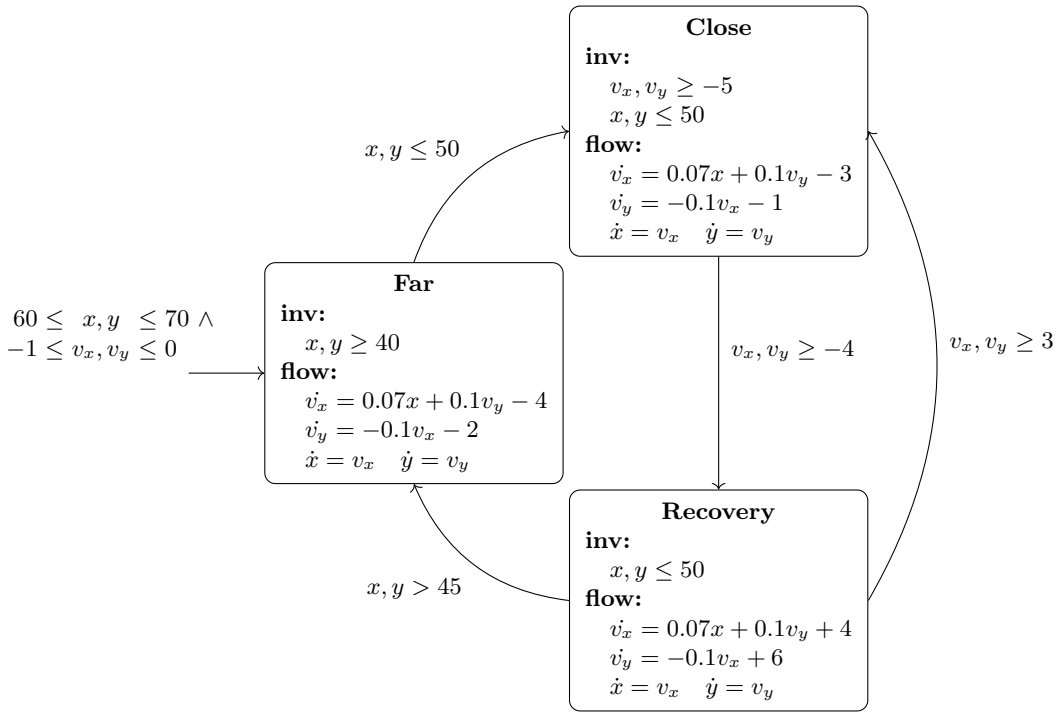


Fig. 13: A hybrid automaton of spacecraft rendezvous maneuver

B.6 Robot Arm

There is a robot arm and a hand is located on the arm. The robot rotates the arm and moves the hand along the arm by an arbitrary speed [2]. The θ is an angle of a robot arm from the ground and and r is a distance of the hand from the center of the robot arm. The ω and v are an angular velocity and the speed of the robot hand, respectively. Figure 15 shows a hybrid automaton H_1 of one battery component. (the other component is similar).

Table 9 shows the STL formulas (for RQ1 and RQ2) used in the experiments.

Figure 16 is a model file of the robot arm controller in STLmc.

B.7 Cardiac Cell

There is standing waves in a cardiac cell, adapted from [15]. The heart contracts for 5 seconds and relaxes for 20 seconds. The cardiac cell expands for 20 seconds and contract for 5 time units. Figure 17 shows a hybrid automaton of the cardiac cell. The membrane potential u, slow gating variable v, and duration d change according to the ODEs in modes, where c is constant in $\mathbb{R}$.

Figure 18 is a model file of the robot arm controller in STLmc.

```
real fx; real fy;
[-200, 200) x; [-200, 200) y;
(-20, 20) vx; (-20, 20) vy;
# far
{
    mode:
        fx = -5; fy = -4 ;
    inv:
        x + y > 100;
    flow:
        d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.7 * x + vy + fx;
        d/dt[vy] = - vx + fy;
    jump:
        (and (x >= -60) (x <= 60)
             (y >= -60) (y <= 60)) =>
        (and (fx' = -3) (fy' = -1)
             (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}
# close
{
    mode:
        fx = -3; fy = -1;
    inv:
        x + y <= 130;
        vx + vy <= 10;
    flow:
        d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.7 * x + vy + fx;
        d/dt[vy] = - vx + fy;
    jump:
        vx + vy >= 8 =>
        (and (fx' = 4) (fy' = 6)
             (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}
# recovery
{
    mode:
        fx = 4; fy = 6;
    inv:
        x + y <= 130;
        vx + vy >= 3;
    flow:
        d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.75 * x + vy + fx;
        d/dt[vy] = - vx + fy;
    jump:
        (or (x > 60) (x < -60)
            (y > 60) (y < -60)) =>
        (and (fx' = -5) (fy' = -4)
             (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
        (vx + vy) <= 5 =>
        (and (fx' = -3) (fy' = -1)
             (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}

init:
    (and (fx = -5) (fy = -4)
         (x <= 150) (x >= 120)
         (y <= 130) (y >= 100)
         (vx <= 0) (vx >= -1)
         (vy <= 0) (vy >= -1));

propositions:
    [vl]: vx + vy <= -5;
    [vr]: vx + vy >= 3;

goal:
    @safe []0, 5] (vl -> (<>[0, 5] vr));
```

Fig. 14: An input model of the spacecraft

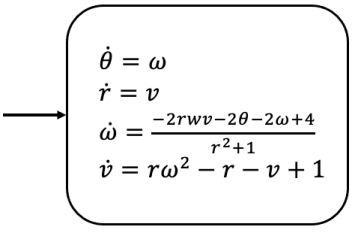


Fig. 15: A hybrid automaton of robot arm

```
bool a;
[-1, 3] theta; [-1, 3] omega;
[-1, 3] r; [-1, 3] v;
{
  mode: a = false;
  inv:  theta < 30; theta > 1;
  flow:
    d/dt[theta] = omega; d/dt[r] = v;
    d/dt[omega] = (- 2 * r * omega * v -
        2 * theta - 2 * omega + 4)/(r * r + 1);
    d/dt[v] = r * omega * omega - r - v + 1;
  jump:
}
```

```
init:
(and (theta >= 1.50) (theta <= 1.51) (r >= 1.50)
   (r <= 1.51) (omega >= 0.0) (omega <= 0.01)
   (v >= 0.0) (v <= 0.00001));

propositions:


#timebound: 15
goal:
@f1: <>[0.0, 4) (rpos -> ([][0.0, 2) vmid));
@f2: [] [0, 5) ((v > 0) -> <> [0, 10) (r > 1));
@f3: <>[0.0, 2.0)([][0, 5) theta <= 2.5);
```

Fig. 16: An input model of the robot arm controller

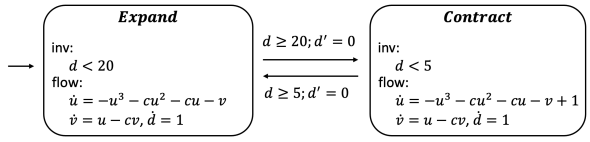


Fig. 17: A hybrid automaton of cardiac cell

```
real modeid; [0, 10] duration;
[-10, 10] u; [-10, 10] v;
{
  mode: modeid = 1;
  inv: duration < 20;
  flow:
    d/dt[duration] = 1;
    d/dt[u] = -  u * u - u * u * u -  u - v;
    d/dt[v] = u - 2 * v;
  jump:
    (duration >= 20) => (and (modeid' = 2)
        (u' = u) (v' = v) (duration' = 0));
}
{
  mode: modeid = 1;
  inv: duration < 5;
  flow:
    d/dt[duration] = 1;
    d/dt[u] = -  u * u - u * u * u -  u - v + 5;
    d/dt[v] = u - 2 * v;
  jump:
    (duration >= 5) => (and (modeid' = 1)
        (u' = u) (v' = v) (duration' = 0));
}
init:
(and (modeid = 2) (duration >= 0)
  (duration <= 0.1) (u >= 0) (u <= 0.1)
  (v >= 0) (v <= 0.1));


propositions:


# timebound: 8
goal:
@f1: ((<>[0.0, 10) (v >= 2)) R[2, 10] (u <= 4));
@f2: <>[1, 3] ((v <= 4) R[2, 5] (u >= 3));
@f3: [][5, 9] ((u <= 4) U[1, 5] (v >= 2));
```

Fig. 18: An input model of a cardiac cell

Table 9: STL for robust STL model checking of robot arm

ID	STL formula	Explanation
φ_1:	$\Diamond_{[0,4)}((r \geq 1.2) \rightarrow \Box_{[0,2)}(v \geq -0.1))$	Within time interval $[0, 4)$, if the distance r is less than or equal to 1.2, than the velocity of the hand v is greater than or equal to -0.1 for time interval $[0, 2)$.
φ_2:	$\Box_{[0,5)}((v > 0) \rightarrow \Diamond_{[0,10)}(r > 1))$	For time interval $[0, 5)$, if v is greater than 0, then r is greater than 1 within time interval $[0, 10)$.
φ_3:	$\Diamond_{[0,2)}(\Box_{[0,5)]}(\theta \leq 2.5))$	Within time interval $[0, 2)$, θ is less than or equal to 2.5 for time interval $[0, 5)$.